第十九章 软件架构技术债

第19章 软件架构技术债

- 19.1 引言
- 19.2 技术债简介
- 19.3 设计债
- 19.4 代码债
- 19.5 测试债
- 19.6 文档债
- 19.7 技术债的处理
- 19.8 小结

19.1引言（1）

- 在实际的软件开发过程中，会遇到下面的这些情况：
  - 在这个开发周期中升级至编译器的新的发行版本已经太晚了，我们决定在下一周期完成这项任务。
  - 我们的开发没有完全按照用户交互接口的规则进行，下个开发周期将做到这点。
  - 我们没有时间去重构小组件的代码，推到下个周期再做。

19.1引言（2）

- 像这种把一些必要的工作推迟进行，或是采用非最优的方案进行，从而保证项目进度正常进行的现象，被称之为一种“欠债”。在软件开发领域称为“技术债（TD, technical debt）”

19.2技术债简介

- 技术债的定义
- 技术债的分类
- 技术债的产生

19.2.1技术债的定义

- 技术债是指开发人员为了加速软件开发，或是由于自身经验的缺乏，有意或无意地在应该采用最佳方案时进行了妥协，使用了短期内能加速软件开发的方案，从而在未来给自己带来的额外开发负担。
- 只要软件发生持续变化，总的技术债就会一直增加，终于有一天会使累积的技术债达到无法偿还的地步，使得不得不放弃该软件产品，这种情况被称为“技术破产”。

19.2.2技术债的分类

- 代码债：开发时没有守代码规范导致的债务，来源包括静态分析工具的违规行为和不一致的编码风格。
- 设计债：在设计时未采用最优解决方案导致的债务，来源包括设计臭味和违背设计规则的行为。架构技术债也是设计债的一种。
- 测试债：测试环节产生的债务，一般由测试的缺乏、测试覆盖面不充分以及不恰当的测试设计导致
- 文档债：由技术文档问题产生的债务，包括缺少重要技术文档、较差的文档和未及时修改更新的文档。

19.2.2技术债的分类

- 技术债的分类及来源

19.2.3技术债的产生（1）

- 产生技术债的一些根本原因：
  - （1）进度压力：当软件开发者接近项目截止日期时，就会努力尽快完成工作，容易仓促行事；
  - （2）软件设计师缺乏足够的经验和技巧：如果软件设计师缺乏对软件设计基础和原则的理解，项目就很容易出现各种问题；
  - （3）不注重设计原则的应用：开发者如果缺乏根据通用原则设计和编写代码的意识或经验，很容易代码难以扩展或修改，尤其是在团队开发或需要不断更新的项目中；

19.2.3技术债的产生（2）

- 产生技术债的一些根本原因：
  - （4）缺乏对设计坏味和重构的意识：很多开发者并不注意设计坏味，而这随着时间的蔓延扩大会导致较差的软件结构质量，产生技术债。
  - （5）开发中有意采用非最优的选择：在某些项目中，有时会有意地做出一些会产生技术债的决定，从而达到某些更重要的目的，同时充分认识到技术债的存在。

19.3设计债

- 设计债的定义
- 设计债的识别方法
- 软件架构技术债

19.3.1 设计债的定义

- 如果在未来还需要为当前软件的结构质量的改进进行付出，就是设计债问题。
- 研究发现，软件设计和架构设计过程中采取的捷径措施及其缺点，都是技术债的组成部分。
- 设计债以前期设计中质量焦点不足的形式存在，或者是以随后缺少重构的零碎设计形式存在。
- 同样，架构技术债可能是因为前期次优的解决方案或因技术和模式的更新而成为次优的解决方案引起的。

19.3.2 设计债的识别方法

- 常见的设计债识别方法有四种：
  - ASA问题识别：自动静态分析工具分析源代码或编译代码来寻找违反推荐设计规程的行为（“问题”），这些问题可能会引起错误或降低软件某些方面的质量（如可维护性、效率）。
  - 代码坏味识别：代码坏味用于描述面向对象系统中违背良好的面向对象设计原则（如信息隐藏、封装、继承的使用）的一些设计。代码坏味道大致可以分为长方法、大类、不稳定口等几十种类型。

19.3.2 设计债的识别方法

- 常见的设计债识别方法有四种：
  - 设计模式污垢识别：随着系统的用途和操作环境的变化，原来的软件设计逐渐发生漂移，这可能波及到设计模式，即原来使用的设计模式随做软件设计变化和漂移，会积累很多污垢（是指与模式无关的代码）；另外，当一些变化破坏了设计模式的结构完整性或功能完整性时，设计模式会发生腐蚀。
  - 模块化冲突：在软件演化过程中，属于不同模块但一起发生变化的组件存在差异。当存在这样的差异时，软件可以偏离它的设计模块结构。

19.3.3 软件架构技术债（1）

- 架构技术债的定义
  - 软件架构技术债是技术债的一种，一般归类为设计债。
  - 架构技术债指的是在软件设计阶段，开发人员为了加速软件开发，在应该采用最佳方案时进行了妥协，改用了短期内能加速软件开发的方案，从而在未来给自己带来的额外开发负担。

19.3.3 软件架构技术债（2）

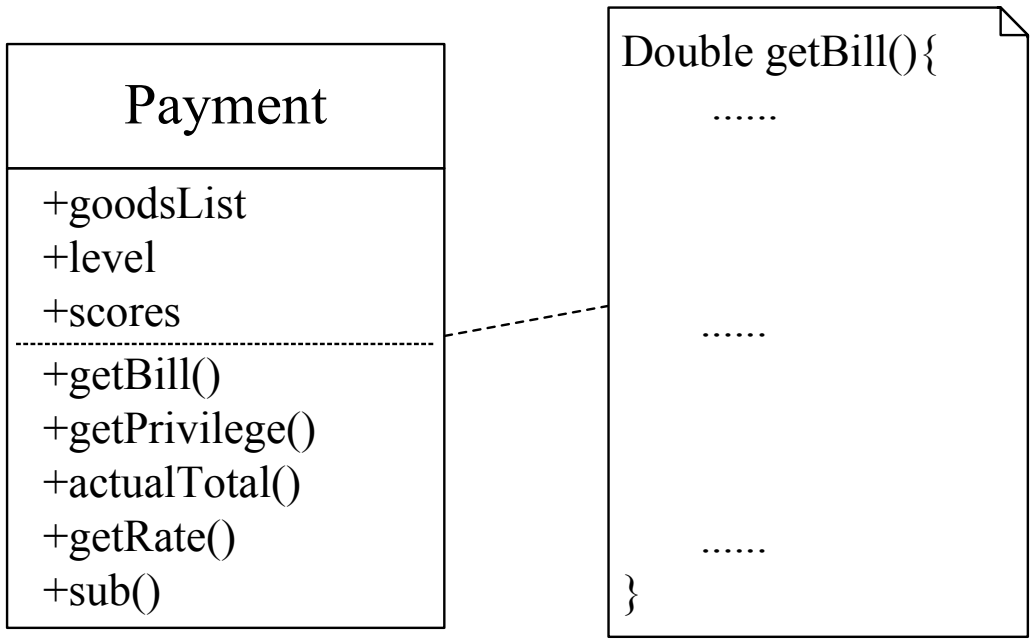
- 架构技术债举例
  - 依赖违背：上下文特定的架构所禁止的架构依赖的存在（如在不同的组件层）被视为软件架构技术债。
  - 模式和策略的不一致性：贯穿整个系统，架构（和架构师）定义的模式和策略可能不会保持一致性。
    - 例如，在系统的某一部分中采用的命名约定可能在系统的另一部分中不会采用。

19.3.3 软件架构技术债（3）

- 架构技术债举例
  - 代码重复(非重用)：文献和实践中公认的是在系统的不同部分中，特别是不同的产品，极其相似（如果不相同）而没有分组到一个重用组件的代码的存在。
  - 相互依存的资源的时序属性：可能需要通过系统的不同部分访问一些资源。
    - 例如，两个组件改变某一数据库的状态或访问该数据库的顺序，可能会改变该系统的行为。
  - 非功能需求的次优机制：一些非功能需求，如可扩展性、性能或信号可靠性，需要在开发过程之前或早期识别出来，并进行测试。

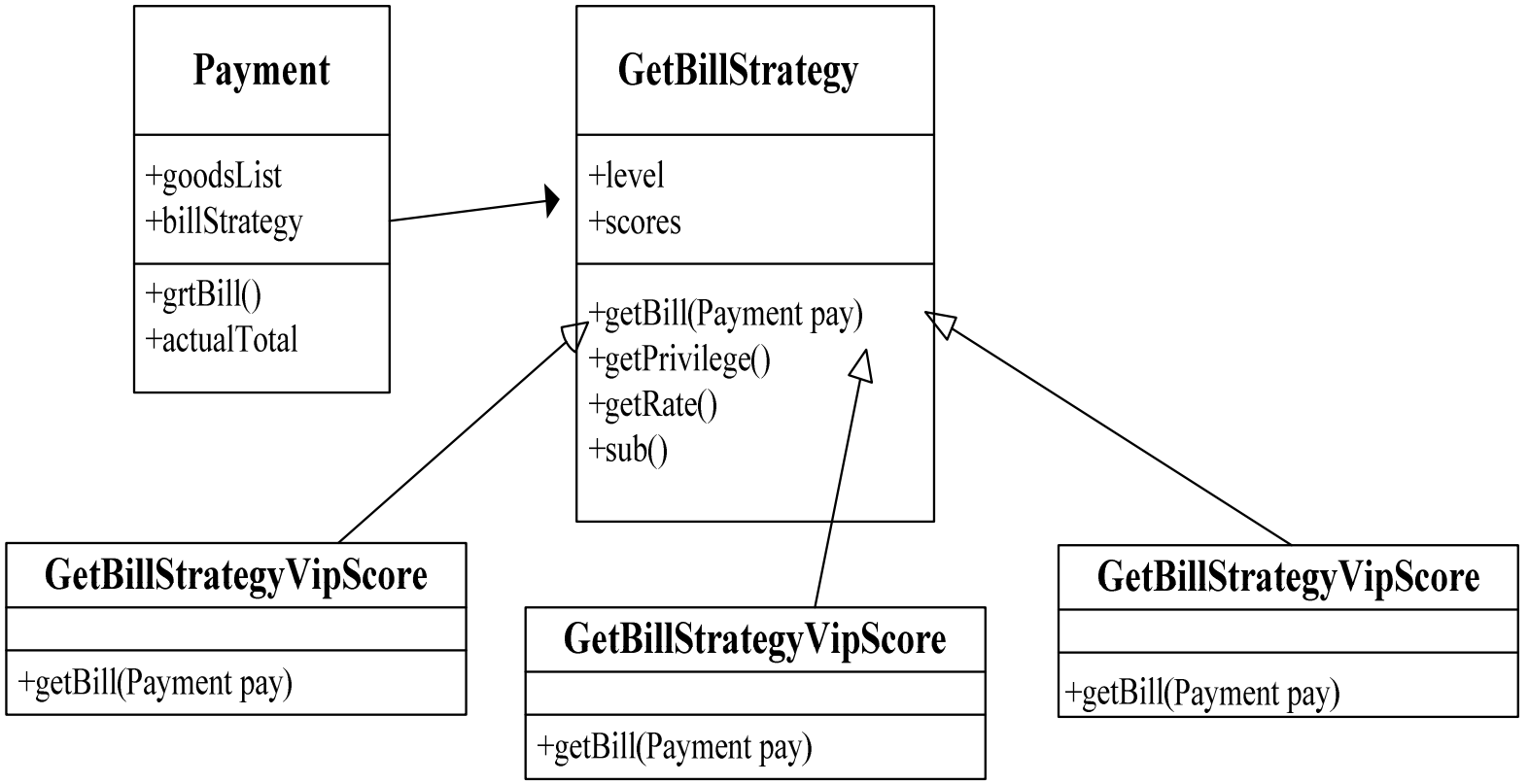
19.3.3 软件架构技术债（4）

- 一个架构技术债的实例
  - 一个商场中用来计算顾客购买商品支付环节的实例，函数getBill()对不同类型的用户进行判断并返回实际应付金额。



19.3.3 软件架构技术债（5）

- 一个架构技术债的实例
  - 采用策略模式进行重构，将客户端的代码与实际算法分离，以便于对算法进行修改或增加新的内容



19.4 代码债

- 代码债
  - 技术债通常会以写的不好的代码为形式出现，这类债务被称为代码债
  - 包括：
    - 不必要的代码重复和复杂性
    - 较低的代码可读性的较差的代码风格
    - 使得软件解决方案在未来某个时间点更新时易于中断的组织不当的逻辑
  - 实质上，需要重构的代码（并且这种重构不属于设计类别的）就是代码债的一种形式。

19.5 测试债

- 测试债
  - 测试债指的是技术债以各种测试脚本问题的形式出现，从而导致每次发布前需要手动重新测试系统，或者无论测试是自动化还是手动运行，测试覆盖面不足、较差的测试设计也是测试债的具体形式。
  - 测试债会在软件开发各个周期的测试阶段造成不利影响，也可能导致客户无法正常维护，损害软件的可用性，进而甚至影响到开发团队整体的品牌和声誉。

19.6 文档债

- 文档债
  - 文档债指的是由技术文档问题产生的债务，包括缺少重要技术文档、较差的文档和未及时修改更新的文档。
  - 一般来说，软件开发的不同环节一个需要撰写十三类技术文档。这些文档应当做到精确、清晰、完整、可追溯。不同的文档缺失、不完善或是更新不及时，都可能导致软件开发或使用遇到问题。

19.7 技术债的处理（1）

- 发现技术债
- 管理技术债
- 偿还技术债

19.7 技术债的处理（2）

- 发现技术债
  - 对于代码类型的技术债，在软件开发、管理和运行中往往会出现一些信号，通过这些信号可以及时找出技术债。
  - 这些信号包括：
    - 系统加载时间变长、特定模块缺陷率不断增加、同一问题在不同的模块或者组件中出现、增加新的功能时，新的bug数量持续增加、修复bug的时间变长、某个模块或者组件难以被团队理解或测试、某个模块或组件的源代码被频繁修改。

19.7 技术债的处理（3）

- 发现技术债
  - 对于非代码类型的技术债（如部分测试债和文档债），更多地是通过提高开发、测试人员的基本技能素质和开发测试规范性来及时发现问题，每个环节都做到精确、清晰、完整、可追溯，从而可以及时发现技术债并进行定位。

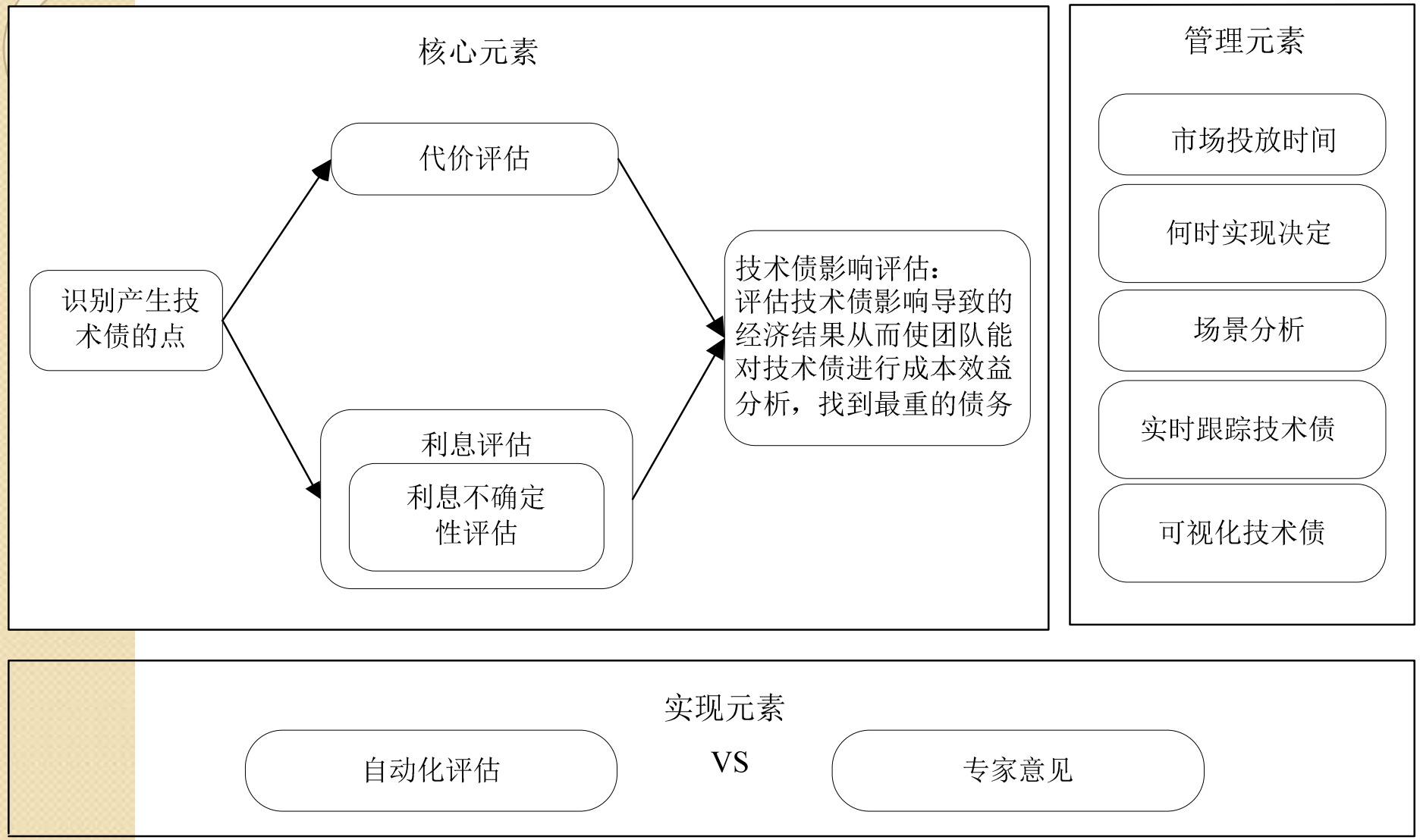
19.7 技术债的处理（4）

- 管理技术债
  - 技术债的管理并不只是软件开发技术的问题，还涉及到项目评估和管理。
  - 技术债管理需要关注的要点：
    - 识别产生技术债的点
    - 代价评估：评估消除特定技术债所花费的代价
    - 利息评估：评估特定技术债未被消除随时间产生的额外代价
    - 利息不确定性评估：评估需要偿还利息的可能概率

19.7 技术债的处理（5）

- 管理技术债
  - 技术债管理需要关注的要点：
    - 技术债影响评估
    - 自动化评估
    - 专家意见
    - 场景分析
    - 市场投放时间：实施决策的时间
    - 何时实现决定
    - 实时跟踪技术债
    - 可视化技术债

技术债管理要点框架


核心元素
识别产生技术债的点
代价评估
利息评估
利息不确定性评估
技术债影响评估：
评估技术债影响导致的经济结果从而使团队能对技术债进行成本效益分析，找到最重的债务
管理元素
市场投放时间
何时实现决定
场景分析
实时跟踪技术债
可视化技术债
实现元素
自动化评估
VS
专家意见

19.7 技术债的处理（6）

- 偿还技术债
  - 技术债的偿还工作是由发现和管理作为基础的，当开发者发现一个技术债时，他并不一定立刻偿还，而是根据技术债的具体情况决定偿还方式。
  - 一般常用的技术债处理方法分为以下四步：
    - （1）发现项目中包含的技术债；
    - （2）将技术债加入到产品列表中；
    - （3）根据债务的影响和偿还成本进行优先级排序；
    - （4）在合适的开发周期中对不同技术债进行修改偿还。

19.8 本章小结

- 在软件开发中，技术债是不可避免的，并且常常可以带来一定的正面收益。
- 放任技术债的存在很可能会给软件带来灾难性的后果，而偿还所有的技术债可能会得不偿失。
- 技术债的偿还应当综合技术代价和实际的收益进行决策，对于软件开发团队来说，技术债的管理是一项意义重大的工作。